REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

Claims 1-30 are currently pending in this application. Claims 1, 7, 13, 20, and 23 are independent. Claims 1-30 were rejected in the Office Action mailed December 30, 2004 ["Action"], in view of different combinations of references. The Applicants respectfully disagree.

I. Amendments

The Applicants have amended claims 1, 7, 13, 20, and 23 for the sake of clarification. No new matter has been added.

II. Cited Art

The Applicants make the following observations in the interest of reaching a shared understanding of the disclosures of Fesslmeier, "C++ Builder 5 Features & Benefits," U.S. Patent No. 6,701,352 to Gardner et al., U.S. Patent No. 6,389,491 to Jacobson et al., and U.S. Patent No. 5,946,489 to Yellin et al.

A. Fesslmeier, "C++ Builder 5 Features & Benefits" ["Fesslmeier"]

Fesslmeier describes various features and benefits of the product C++ Builder 5. C++ Builder 5 is a C++ development system. The system includes authoring tools. The system also includes C++ compiler technology. [*See, e.g.*, pages 7-8 – "3 The most powerful ANSI/ISO C++ compiler."]

On page 9, Fesslmeier describes features to "Simplify CORBA Distributed Object Development." One feature cited in the Action is "IDL Integration," in which:

> Integrated IDL simplifies CORBA development by automating IDL compilation and implementation generation. As interface definitions change, C++ implementations are *automatically kept in sync in both Client and Server projects*. IDL Syntax highlighting affirms coding and reduces errors when writing and updating interface definitions. [Fesslmeier, page 9, emphasis added.]

The Applicants make two observations about this section.

First, in this section, Fesslmeier describes automatically synchronizing C++ implementation code for both client and server projects when an interface definition changes. This updating of C++ code appears to be done during development by an authoring tool, not during compile time. As such, *the product of this updating process is C++ code, not executable code.* Because the described process produces C++ during authoring, the process of Fesslmeier must be done prior to any compilation of the updated C++ code. Additionally, *this section does not describe the parsing of IDL during compilation or the generation of any executable code using IDL information during compilation.*

Second, the Applicants understand the phrase "IDL compilation" to refer to the process of creating IDL stubs and skeletons for an object. This is consistent with the CORBA specification by Object Management Group, *The Common Object Request Broker: Architecture and Specification, revision 2.3*, June 1999, available online at *http://www.omg.org/technology/documents/vault.htm*. ["OMG"] OMG, in figures 2-2, 2-3, 2-4, and 2-5 illustrates "IDL stubs" and "static IDL skeletons." Additionally, OMG, at page 2-5 describes an interface definition defined in IDL as "used to generate the client Stubs and the object implementation skeletons." The IDL stubs and skeletons are used for static procedure calls in the CORBA distributed object system.

On page 11, Fesslmeier describes tools for creating components directly out of COM+ servers. The created components then behave like other components in the C++Builder 5 environment. One passage cited in the Action, "Integrated Type Library creation reduces the number of programming tasks," describes benefits of integrated type library creation:

> Using and creating IDL (Interface Definition Langauge) has never been easier. Because the TypeLibrary Editor is integrated into COM and ActiveX development, the developer has the ability to *change things in code, in IDL, or graphically and always have a synchronized view into complex COM development.* This will *save hours of programming frustration* and will flatten the COM+/ActiveX learning curve.
> [Fesslmeier, page 11, emphasis added.]

The Applicants again note that the passage describes synchronization between code and IDL. Additionally, this synchronization appears to be done during development as well, and not at compile time by a compiler. As such, the passage also *does not describe the parsing of IDL and programming code to create executable code during compilation.*

Fesslmeier also describes a C++ compiler. [Fesslmeier, page 7] The C++ compiler does not parse IDL when generating an interface implementation or otherwise creating executable code.

B. U.S. Patent No. 6,701,352 to Gardner et al. ["Gardner"]

Gardner describes various aspects of an Object Linking and Embedding Automation facility in Microsoft Windows 95. It describes this as an example of "bridge software" which "enables one application program to communicate with another application program through an agreed-upon, shared inter-application communication scheme." [Gardner, 7:46-50.] It further describes bridge software during *runtime operation* of applications:

> In one embodiment, the bridge software 210 provides a distributed object communication facility. An application program can write data to an object, and invoke a transport routine to cause the bridge software 210 to transport the object to another application program. The receiving application program extracts a message and data from the object and acts upon them. [Gardner, 7:50-56.]

With regard to OLE in particular, Gardner describes the use of "dispatchable interfaces" and "late binding":

> Using dynamic dispatching under OLE, an automation client can invoke a method or manipulate a property of a server component by a late binding mechanism. *At run time, the automation client obtains a dispatch identifier from a type library* associated with the server component. The dispatch identifier is passed to an "invoke" method of OLE Automation that resolves which method of the server component to call at run time. *The type library is created at run time* by an object definition language (ODL) file that describes interfaces of the server component. [Gardner, 8:4-14, emphasis added.]

While cited passages of Gardner discuss runtime use of dispatchable interfaces and late binding (e.g., using the Invoke method), they do not does not discuss generation of interface implementations. What they specifically *do not* describe is any method of generating dispatch interface implementations, whether from definition information or otherwise. It is also worth noting that Gardner does not describe specifics of any particular interfaces, but rather methods of late binding to the particular interfaces using type libraries at run time, where the late binding methods are described by definition language.

Finally, Gardner discusses the alternate use of the Common Object Request Broker Architecture as the bridge software 210. [Gardner, 8:20-22.]

C. U.S. Patent No. 5,946,489 to Yellin et al. ["Yellin"]

Yellin describes an example of placing code of one type within a file of a different type of code, using inlining. Yellin describes the use of instructions in C code that generate assembly code when compiled. [Yellin, 8:17-24.]

D. U.S. Patent No. 6,389,491 to Jacobson et al. ["Jacobson"]

Jacobson describes the use of a dual interface mechanism:

> As known by those skilled in the art, ActiveX Automation Servers by definition support dual interfaces, including custom vtable-based access interfaces such as the IIO interface 38b and the well-known IDispatch interface which supports method invocation for applications written in languages that do not support vtable access. Thus, implementing the universal I/O interface 30 as an ActiveX Universal I/O Automation Server allows any language that can support vtable access to access the methods directly via the vtable. Languages that support late binding can invoke the methods via the Invoke( ) method of the IDispatch interface. [Jacobson, 4:65-5:9.]

Thus, Jacobson describes both direct access to interface methods using the vtable and late-bound access using IDispatch interface.

III. Claims 1, 3, 4, 6, and 26

Claim 1 is directed to generating a dispatch interface implementation. Definition information is received which defines dispatch interface features of a dispatch interface that includes dispatch methods. A dispatch interface implementation for operating one or more other methods is generated, where the generating includes parsing the definition information as well as the programming language code (for the one or more other methods) during compilation. The implementation includes executable code for: (1) the one or more other methods, (2) a dispatch method for mapping names (of the one or more other methods) to dispatch identifiers for binding at run time, and (3) a dispatch method for calling the one or more other methods at run time responsive to client requests.

The Action rejects claims 1, 3, 4, 6, and 26 as being unpatentable over Fesslmeier in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 1 should be allowable.

A. Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 1.

Claim 1 recites at least one limitation that Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 1 recites "generating a dispatch interface implementation for operating the one or more methods, wherein the generating includes parsing the programming language code and the definition information during compilation." The portions of Fesslmeier (pages 9 and 11) cited against the above-cited language involve updating C++ code based upon IDL changes *using an authoring tool*. Fesslmeier does not anywhere teach or suggest generating a dispatch interface implementation, where the generating includes parsing programming language code and definition information during compilation. In fact, as mentioned above, the cited passages of Fesslmeier describe a process using IDL information that must take place *before* compilation and which *does not produce executable code*.

Gardner, in turn, does not relate to generation of a dispatch interface implementation or any activities during compilation, including parsing of programming language code or definition information, and is even further from teaching or suggesting generation during compilation of a dispatch interface, where that generation includes parsing definition information and programming language code, as recited in claim 1.

Because Fesslmeier and Gardner taken separately fail to teach or suggest the above-cited language of claim 1, the combination of Fesslmeier and Gardner also fails to teach or suggest the above-cited language of claim 1, and claim 1 should be allowable.

B. The combination of Fesslmeier and Gardner is improper.

The combination proposed by the Examiner to reject claim 1 is improper. The Examiner admits that Fesslmeier does not disclose "dispatch interface from definition information." [Action, page 3.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "dispatch interface" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement the automated system of C++ Builder with late binding as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace. [*Id.*]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier and Gardner that lead away from making the modification suggested by the Examiner.

Fesslmeier describes development tools for using definition information to automatically *synchronize C++* implementation code for both client and server projects when an interface definition changes during development. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The Examiner proposes modifying Fesslmeier according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time.* [Gardner 8:7-14.] According to late binding as in Gardner a client and server *need not be synchronized* to the same interface definition before they begin to interoperate.

Further, Gardner describes the use of definition information at runtime to create a type library. [Gardner 8:12-14.] This also changes the principle of operation of Fesslmeier, which uses definition information prior to compilation. Using Gardner to modify Fesslmeier (as the Examiner has done) goes against the principles of operation of both Gardner and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a remote procedure call ["RPC"]. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of dispatch interfaces is to facilitate run time negotiation concerning layout and/or makeup of

methods of an interface. (A dispatch interface allows in some embodiments, for example, changing of the interface by adding or deleting methods and properties after the interface is "published," such that a client and server may interoperate across the interface even though the client and server are not "in sync" as to the current definition of the interface when they begin the interoperation. See Al Major, COM IDL & Interface Design, Wrox Press, pp. 103-104, 1999 (previously cited in an Information Disclosure Statement).) While this argument was made in a previous Amendment, the Action does not address it.

For at least these reasons, claim 1 should be allowable.

Claims 3, 4, 6, and 26 (which depend from claim 1) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 3, 4, 6, and 26.

IV. Claim 2

The Action rejects claim 2 (which depends from claim 1) as being unpatentable over Fesslmeier in view of Gardner and Yellin. The Applicants respectfully disagree.

Taken separately or in combination with the other references, Yellin does not teach or suggest the above-cited language of claim 1 that Fesslmeier and Gardner fail to teach or suggest (see section III). Moreover, the combination of Fesslmeier with Gardner and Yellin is improper for at least the reasons that the combination of Fesslmeier with Gardner is improper (see section III).

The Applicants will not belabor the merits of the separate patentability of claim 2. Claim 2 should be allowable.

V. Claim 5

The Action rejects claim 5 (which depends from claim 1) as being unpatentable over Fesslmeier in view of Gardner and Jacobson. The Applicants respectfully disagree.

Taken separately or in combination with the other references, Jacobson does not teach or suggest the above-cited language of claim 1 that Fesslmeier and Gardner fail to teach or suggest (see section III). Moreover, the combination of Fesslmeier with Gardner and Jacobson is improper for at least the reasons that the combination of Fesslmeier with Gardner is improper (see section III).

The Applicants will not belabor the merits of the separate patentability of claim 5. Claim 5 should be allowable.

VI. Claims 7-12 and 27

Claim 7 is directed to a compiler system that generates a late binding interface implementation. The compiler system includes a front end module, a converter module, and a back end module. The front end module receives definition information (defining late binding interface features of a late binding interface) and programming language code (for implementing one or more late bound methods). The converter module identifies relations between the definition information and the one or more late bound methods during compilation, including parsing the definition information and the programming language code. The back end module generates a late binding interface implementation based upon the relations, for operating the one or more late bound methods.

The Action rejects claims 7-12 and 27 as being unpatentable over Fesslmeier in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 7 should be allowable.

A. Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 7.

Claim 7 recites at least one limitation that Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 7 recites a "compiler system" that includes a "converter module that identifies relations between the definition information and the one or more late bound methods during compilation, including parsing the definition information and the programming language code" and "a back end module that generates a late binding interface implementation based upon the relations." The portions of Fesslmeier (pages 9 and 11) cited against the above-cited language involve updating C++ code based upon IDL changes *using an authoring tool.* Fesslmeier does not anywhere teach or suggest a compiler system that identifies relations between definition language and late bound methods *during compilation (including parsing definition information and programming language code)* and generates an interface implementation based upon the relations. In fact, as mentioned above, the cited

passages of Fesslmeier describe a process using IDL information that must take place *before* compilation and which *does not produce executable code.*

Gardner, in turn, does not relate to compiler systems or parsing of programming language code or definition information, and is even further from teaching or suggesting a compiler system that parses definition information as recited in claim 7.

Because Fesslmeier and Gardner taken separately fail to teach or suggest the above-cited language of claim 7, the combination of Fesslmeier and Gardner also fails to teach or suggest the above-cited language of claim 7, and claim 7 should be allowable.

B. The combination of Fesslmeier and Gardner is improper.

The combination proposed by the Examiner to reject claim 7 is improper. The Examiner admits that Fesslmeier does not disclose "late bound." [Action, page 5.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "dynamic/late binding, dispatchable" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement the automated system of C++ Builder with late binding as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace. Furthermore, both references involve CORBA object linking. [Action, pages 5-6.]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier and Gardner that lead away from making the modification suggested by the Examiner.

Fesslmeier describes development tools for using definition information to automatically *synchronize* C++ implementation code for both client and server projects when an interface definition changes during development. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The Examiner proposes modifying Fesslmeier according to Gardner, so as to allow a client and server to communicate by

using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time.* [Gardner 8:7-14.] According to late binding as in Gardner a client and server *need not be synchronized* to the same interface definition before they begin to interoperate.

Further, Gardner describes the use of definition information at runtime to create a type library. [Gardner 8:12-14.] This also changes the principle of operation of Fesslmeier, which uses definition information prior to compilation. Using Gardner to modify Fesslmeier (as the Examiner has done) goes against the principles of operation of both Gardner and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a remote procedure call ["RPC"]. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of dispatch interfaces is to facilitate run time negotiation concerning layout and/or makeup of methods of an interface. (A dispatch interface allows in some embodiments, for example, changing of the interface by adding or deleting methods and properties after the interface is "published," such that a client and server may interoperate across the interface even though the client and server are not "in sync" as to the current definition of the interface when they begin the interoperation. See Al Major, COM IDL & Interface Design, Wrox Press, pp. 103-104, 1999 (previously cited in an Information Disclosure Statement).) While this argument was made in a previous Amendment, the Action does not address it.

For at least these reasons, claim 7 should be allowable.

Claims 8-12 and 27 (which depend from claim 7) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 8-12 and 27.

VII. Claims 13-18 and 28

Claim 13 is directed to generating a late binding interface implementation. Definition information defines late binding interface features of a late binding interface. A late binding interface implementation for operating one or more late bound methods is generated during compilation, where the generating includes parsing programming language code (for the one or

more late bound methods) and definition information. The implementation includes one or more late binding methods, including a method for calling the one or more late bound methods responsive to client requests.

The Action rejects claims 13-18 and 28 as being unpatentable over Fesslmeier in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 13 should be allowable.

For at least the following reasons, claim 13 should be allowable.

A. Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 13.

Claim 13 recites at least one limitation that Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 13 recites "generating a late binding interface implementation for operating the one or more late bound methods, wherein the generating includes parsing the programming language code and the definition information during compilation." The portions of Fesslmeier (pages 9 and 11) cited against the above-cited language involve updating C++ code based upon IDL changes *using an authoring tool*. Fesslmeier does not anywhere teach or suggest generating a late binding interface implementation, where the generating includes parsing programming language code and definition information during compilation. In fact, as mentioned above, the cited passages of Fesslmeier describe a process using IDL information that must take place *before* compilation and which *does not produce executable code*.

Gardner, in turn, does not relate to generation of a late binding interface implementation with a compiler system, where that generation includes parsing of programming language code or definition information, and is even further from teaching or suggesting a compiler system that parses definition information as recited in claim 13.

Because Fesslmeier and Gardner taken separately fail to teach or suggest the above-cited language of claim 13, the combination of Fesslmeier and Gardner also fails to teach or suggest the above-cited language of claim 13, and claim 13 should be allowable.

B. The combination of Fesslmeier and Gardner is improper.

The combination proposed by the Examiner to reject claim 13 is improper. The Examiner admits that Fesslmeier does not disclose various "late bound" language in claim 13. [Action, page 8.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates the "late bound" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement the automated system of C++ Builder with late binding as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace." [*Id.*]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier and Gardner that lead away from making the modification suggested by the Examiner.

Fesslmeier describes development tools for using definition information to automatically *synchronize C++* implementation code for both client and server projects when an interface definition changes during development. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The Examiner proposes modifying Fesslmeier according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time.* [Gardner 8:7-14.] According to late binding as in Gardner a client and server *need not be synchronized* to the same interface definition before they begin to interoperate.

Further, Gardner describes the use of definition information at runtime to create a type library. [Gardner 8:12-14.] This also changes the principle of operation of Fesslmeier, which uses definition information prior to compilation. Using Gardner to modify Fesslmeier (as the Examiner has done) goes against the principles of operation of both Gardner and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of

simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a remote procedure call ["RPC"]. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of dispatch interfaces is to facilitate run time negotiation concerning layout and/or makeup of methods of an interface. (A dispatch interface allows in some embodiments, for example, changing of the interface by adding or deleting methods and properties after the interface is "published," such that a client and server may interoperate across the interface even though the client and server are not "in sync" as to the current definition of the interface when they begin the interoperation. See Al Major, COM IDL & Interface Design, Wrox Press, pp. 103-104, 1999 (previously cited in an Information Disclosure Statement).) While this argument was made in a previous Amendment, the Action does not address it.

For at least these reasons, claim 13 should be allowable.

Claims 14-18 and 28 (which depend from claim 13) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 14-18 and 28.

VIII. Claim 19

The Action rejects claim 19 (which depends from claim 13) as being unpatentable over Fesslmeier in view of Gardner and Jacobson. The Applicants respectfully disagree.

Taken separately or in combination with the other references, Jacobson does not teach or suggest the above-cited language of claim 13 that Fesslmeier and Gardner fail to teach or suggest (see section VII). Moreover, the combination of Fesslmeier with Gardner and Jacobson is improper for at least the reasons that the combination of Fesslmeier with Gardner is improper (see section VII).

The Applicants will not belabor the merits of the separate patentability of claim 19. Claim 19 should be allowable.

IX. Claims 20-22 and 29

Claim 20 is directed to automatically generating an interface implementation having early binding and late binding mechanisms. Definition information defines late binding interface features of the interface. An interface implementation is generated at compile time for

alternatively operating one or more methods by an early binding mechanism or by a late binding mechanism, where the generating includes parsing the definition information and programming language code (for the one or more methods of the interface). The early binding mechanism provides for direct invocation of the one or more methods. (See, e.g., page 3 of the application, which describes the use of "virtual function tables" in early binding of method names to method code in an interface.) The late binding mechanism provides for invocation of the one or more methods responsive to a request through a late binding method.

The Action rejects claims 20-22 and 29 as being unpatentable over Fesslmeier in view of Gardner and Jacobson. The Applicants respectfully disagree. For at least the following reasons, claim 20 should be allowable.

A. Fesslmeier, Gardner, and Jacobson, taken separately or in combination, fail to teach or suggest at least one limitation of claim 20.

Claim 20 recites at least one limitation that Fesslmeier, Gardner, and Jacobson, taken separately or in combination, fail to teach or suggest. For example, claim 20 recites "generating an interface implementation for alternatively operating the one or more methods by an early binding mechanism or by a late binding mechanism, wherein the generating includes parsing the programming language code and the definition information during compilation." The portions of Fesslmeier (pages 9 and 11) cited against the above-cited language involve updating C++ code based upon IDL changes *using an authoring tool*. Fesslmeier does not anywhere teach or suggest generating an interface implementation for alternately using an early or late binding mechanism, where the generating includes parsing definition information and programming language code during compilation. In fact, as mentioned above, the cited passages of Fesslmeier describe a process using IDL information that must take place *before* compilation and which *does not produce executable code*.

Gardner, in turn, does not relate to generation of an interface implementation or any compile time activities, or parsing of programming language code or definition information, and is even further from teaching or suggesting the above-cited language of claim 20. Likewise, Jacobson does not relate to generation of an interface implementation or compile time activities, and is even further from teaching or suggesting the above-cited language of claim 20.

Because Fesslmeier, Gardner, and Jacobson taken separately fail to teach or suggest the above-cited language of claim 20, the combination of Fesslmeier, Gardner, and Jacobson also fails to teach or suggest the above-cited language of claim 20, and claim 20 should be allowable.

B. The combination of Fesslmeier, Gardner, and Jacobson is improper.

The combination proposed by the Examiner to reject claim 20 is improper. The Examiner admits that Fesslmeier does not disclose various "late bound" language in claim 20. [Action, page 18.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "late bound" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement the automated system of C++ Builder with late binding as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide object interface for as many environments as possible in order to facilitate greater acceptance and use in the marketplace. Furthermore, both reference involve CORBA object linking" [*Id.*]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier and Gardner that lead away from making the modification suggested by the Examiner.

Fesslmeier describes development tools for using definition information to automatically *synchronize C++* implementation code for both client and server projects when an interface definition changes during development. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The Examiner proposes modifying Fesslmeier according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time.* [Gardner 8:7-14.] According to late binding as in Gardner a client and server *need not be synchronized* to the same interface definition before they begin to interoperate.

Further, Gardner describes the use of definition information at runtime to create a type library. [Gardner 8:12-14.] This also changes the principle of operation of Fesslmeier, which uses definition information prior to compilation. Using Gardner to modify Fesslmeier (as the Examiner has done) goes against the principles of operation of both Gardner and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a remote procedure call ["RPC"]. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of dispatch interfaces is to facilitate run time negotiation concerning layout and/or makeup of methods of an interface. (A dispatch interface allows in some embodiments, for example, changing of the interface by adding or deleting methods and properties after the interface is "published," such that a client and server may interoperate across the interface even though the client and server are not "in sync" as to the current definition of the interface when they begin the interoperation. See Al Major, COM IDL & Interface Design, Wrox Press, pp. 103-104, 1999 (previously cited in an Information Disclosure Statement).) While this argument was made in a previous Amendment, the Action does not address it.

The combination of Fesslmeier with Gardner and Jacobson is improper for at least the reasons that the combination of Fesslmeier with Gardner is improper.

For at least these reasons, claim 20 should be allowable.

Claims 21-22 and 29 (which depend from claim 20) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 21-22 and 29.

X. Claims 23-25 and 30

Claim 23 is directed to automatically generating call site code for calling a late bound method through a late binding interface. Received definition information and programming language code (for calling a late bound method) are parsed. Based upon type information for one or more input arguments of the late bound method, code is generated for packing the one or more input arguments into a generic argument data structure. Code is also generated for calling

the late bound method through an invocation method of the late binding interface, wherein the calling includes passing the generic argument data structure to the invocation method.

The Action rejects claims 23-25 and 30 as being unpatentable over Fesslmeier in view of Gardner. The Applicants respectfully disagree. For at least the following reasons, claim 23 should be allowable.

A. Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest at least one limitation of claim 23.

Claim 23 recites at least one limitation that Fesslmeier and Gardner, taken separately or in combination, fail to teach or suggest. For example, claim 23 recites "parsing the definition information and the programming language code during compilation." The portions of Fesslmeier (pages 9 and 11) cited against the above-cited language involve updating C++ code based upon IDL changes *using an authoring tool*. Fesslmeier does not anywhere teach or suggest parsing definition information and programming language code during compilation. In fact, as mentioned above, the cited passages of Fesslmeier describe a process using IDL information that must take place *before* compilation and which *does not produce executable code*.

Gardner, in turn, does not relate to parsing of programming language code or definition information, and is even further from teaching or suggesting parsing programming language code and definition information during compilation, as recited in claim 23.

Because Fesslmeier and Gardner taken separately fail to teach or suggest the above-cited language of claim 23, the combination of Fesslmeier and Gardner also fails to teach or suggest the above-cited language of claim 23, and claim 23 should be allowable.

B. The combination of Fesslmeier and Gardner is improper.

The combination proposed by the Examiner to reject claim 23 is improper. The Examiner admits that Fesslmeier does not disclose "dispatch interface from definition information." [Action, page 11.] The Applicants agree. The Examiner argues, however, that Gardner demonstrates "dispatch interface" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement the automated system of C++ Builder

> with late binding as found in Gardner's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to develop interfaces for as many situations as possible and thus improve competitiveness and user convenience, including late bound, expecially when those interfaces are needed for so many applicaitons. [*Id.*]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier and Gardner that lead away from making the modification suggested by the Examiner.

Fesslmeier describes development tools for using definition information to automatically *synchronize C++* implementation code for both client and server projects when an interface definition changes during development. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The Examiner proposes modifying Fesslmeier according to Gardner, so as to allow a client and server to communicate by using dispatchable interfaces for which interface information is *obtained at run time from a type library that is not created until run time*. [Gardner 8:7-14.] According to late binding as in Gardner a client and server *need not be synchronized* to the same interface definition before they begin to interoperate.

Further, Gardner describes the use of definition information at runtime to create a type library. [Gardner 8:12-14.] This also changes the principle of operation of Fesslmeier, which uses definition information prior to compilation. Using Gardner to modify Fesslmeier (as the Examiner has done) goes against the principles of operation of both Gardner and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a remote procedure call ["RPC"]. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of dispatch interfaces is to facilitate run time negotiation concerning layout and/or makeup of methods of an interface. (A dispatch interface allows in some embodiments, for example,

changing of the interface by adding or deleting methods and properties after the interface is "published," such that a client and server may interoperate across the interface even though the client and server are not "in sync" as to the current definition of the interface when they begin the interoperation. See Al Major, COM IDL & Interface Design, Wrox Press, pp. 103-104, 1999 (previously cited in an Information Disclosure Statement).) While this argument was made in a previous Amendment, the Action does not address it.

For at least these reasons, claim 23 should be allowable.

Claims 24, 25, and 30 (which depend from claim 23) should also be allowable, but the Applicants will not belabor the merits of the separate patentability of claims 24, 25, and 30.

CONCLUSION

Claims 1-30 should be allowed. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By _____
Kyle B. Rinehart
Registration No. 47,027

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446